OS Project Report
Ludo
22i-1552, 22i-1560, 21i-2775

Submitted To:

Sir Arsalan Aslam

15th May, 2024

Pseudocode:

//First of all include all header files eg: pthread.h, semaphore.h, and other .h files that we need

```
//Declare global variables

const dicet = [1, 6, 2, 3, 4, 5, 6, 6]  // Array of dice values
dice = [0, 0, 0]                       // Array to hold dice rolls
plays = 0                              // Number of plays
gotein = 0                             // Game state variable

/// setting the color codes for displaying

RESET_CLR = "reset color"
RED_CLR = "red color"
GRN_CLR = "green color"
YELO_CLR = "yellow color"
BLU_CLR = "blue color"
RED_BG = "red background color"
GRN_BG = "green background color"
YELO_BG = "yellow background color"
BLU_BG = "blue background color"
flash = "flash effect"


// declare array for board and player tracks
track = array of size 52
ltrack = 2D array of size 4x5


// Function to reset track element
function trackreset(i)
   track[i] = "reset track element to default color and value"

// Function to reset ltrack element
function ltrackreset(i, j)
   ltrack[i][j] = "reset ltrack element to default color"

// Function to clear console screen
function cls()
   print("clear screen command")

// Function to wait for specified time
function plswait(k)
```

```
    calculate time in seconds and nanoseconds from k
    sleep for calculated time

// Define Player class
class Player {
    // Player attributes
    bool beat = false
    bool active = true
    string oout
    int id
    int tokco
    int offset
    array prog
    array pos

    // Constructor
    Player() {
            id = pcount  // Set player ID to current count (and increment count)
            pcount += 1  // Increment player count for next player creation
            tokco = gotein  // Set number of tokens per player
            oout = "\e[0;1;91m#\e[0m"  // Initialize token representation (formatted string)

            // Create arrays to track token progress and positions
            prog = new int[tokco]  // Array to track token progress
            pos = new int[tokco]  // Array to track token positions

            // Initialize token progress and positions
            for i from 0 to tokco - 1:
                prog[i] = -1  // Set token progress to -1 (not on the board)
                pos[i] = -1   // Set token position to -1 (not on the board)

            // Adjust offset based on player ID for board positioning
            offset = 52  // Assuming default offset value (initial board position)
            offset -= (id + 1) * 13  // Adjust offset based on player ID

            // Additional actions (e.g., input player name)
            // Note: Uncomment and modify as needed based on specific game logic
            // printf("Enter player %i name:", id)
            // cin >> nam
            // cout << oout << endl

            // End of constructor
    }

    // Method to check if any token can be moved
    function canMove(n) {
                x = -1  // Initialize to no valid move
```

```
// Check if no tokens are open and not a roll of 6
if (not anyOpen() and n != 6):
    return x  // No tokens can move

// Iterate through each token
for i from 0 to tokco - 1:
    // Check if token is valid to move if roll is a 6
    if (prog[i] > -1 and n == 6):
        x = i  // Token can move

    // Check if token can legally move to certain positions
    if ((prog[i] + n) != 13 and (prog[i] + n) != 21 and (prog[i] + n) != 26 and
        (prog[i] + n) != 34 and (prog[i] + n) != 39 and (prog[i] + n) != 47):

        // Check if there is an opponent's token to wreck
        if (beat):
            // Check special conditions for home and non-overlapping with friends
            if ((prog[i] + n) == 8 or (prog[i] == 51 and n == 1)):
                x = i
                continue  // Continue to next token

            // Check for overlap with opponent tokens
            for j from 0 to tokco - 1:
                if (prog[i] != 51):
                    // Check if not overlapping with friendly tokens
                    if (i != j and (prog[i] + n != prog[j])):
                        x = i
                        // Check if opponent is on the track
                        if (prog[i] + n < 51):
                            if (track[(pos[i] + n) % 52][2] != '4'):
                                x += 4  // Opponent on the track
                else:
                    // Check if not overlapping with friendly tokens
                    if (i != j and ((prog[i] + n) % 52) != prog[j]):
                        x = i
                        // Check if opponent is on the track
                        if (track[(pos[i] + n) % 52][2] != '4'):
                            x += 4  // Opponent on the track

        else:
            // No beat yet, check standard movement conditions
            if ((prog[i] + n) == 8 or (prog[i] == 51 and n == 1)):
                x = i
                continue  // Continue to next token

            // Check for overlap with opponent tokens
            for j from 0 to tokco - 1:
                // Check if not overlapping with friendly tokens
```

```
                    if (i != j and ((prog[i] + n) % 52) != prog[j]):
                        x = i
                        // Check if opponent is on the track
                        if (track[(pos[i] + n) % 52][2] != '4'):
                            x += 4  // Opponent on the track

            return x  // Return the index of the token that can move
}

// Method to bury a token at a given position
function bury(p) {
    for i from 0 to tokco - 1:
    // Check if the current token is at the specified position (p)
    if pos[i] == p:
        // "Bury" the token by resetting its progress
        prog[i] = -1  // Set token progress to -1 (not on the board)

        // Update the track to show a flash (indicating token removal)
        track[p] = flash

        // Display the updated game board after token removal
        displayBoard()

        // Reset the track at the token's position
        trackreset(p)

        // Display the updated game board after track reset
        displayBoard()

        // Reset the token's position to indicate it's no longer on the board
        pos[i] = -1  // Set token position to -1 (not on the board)

}

// Method to move a token
function move(i, n) {
                if i < 4:
    // Check if the token is a valid index within the player's tokens
    if prog[i] < 0 and n == 6:
        // Token is not on the board and a 6 is rolled, so place it on the board
        print("Token opened")
        prog[i] = 0
        pos[i] = offset
        track[pos[i]] = oout
        Return

        else if beat:
                // Token is on the board and a beat has occurred (opponent token being
        wrecked)
                if prog[i] < 51:
                // Token is not at the final position yet
                trackreset(pos[i])
                if (prog[i] + n) > 50:
                    // Token will reach or pass the final position
                    prog[i] += (n + 1)
                    pos[i] = prog[i] % 52
                    if prog[i] == 57:
                        return  // Token reached the end
                    ltrack[id][pos[i]] = oout
                else:
                    // Token moves normally within the track
                    prog[i] += n
                    pos[i] += n
                    pos[i] %= 52
                    track[pos[i]] = oout
                else if prog[i] == 51:
                // Token is exactly at the final position
                trackreset(pos[i])
                prog[i] = n - 1
                pos[i] = offset + (n - 1)
                track[pos[i]] = oout
                else:
                // Token is in the leaving track
                ltrackreset(pos[i])
                prog[i] += n
                pos[i] += n
                if prog[i] == 57:
                    return  // Token reached the end
                ltrack[id][pos[i]] = oout

        else:
                // Normal movement when no beat has occurred
                trackreset(pos[i])
                prog[i] += n
                prog[i] %= 52
                pos[i] += n
                pos[i] %= 52
                track[pos[i]] = oout

            else:
        // Token index is not within the player's tokens (used for wrecking opponents)
        trackreset(pos[i])
        o = track[(pos[i] + n) % 52][7] - '1'
        ((player *)p)[o].bury((pos[i] + n) % 52)
        prog[i] += n
```

```
                pos[i] += n
                pos[i] %= 52
                beat = 1
        }
}
```

Function anyopen():

```
  - Initialize a boolean variable x to false
  - Loop through each token (from 0 to tokco - 1)
    - If prog[i] is between 0 and 57 (inclusive)
      - Set x to true
      - Break the loop
  - Return x
```

Function anyclose():

```
  - Initialize a boolean variable x to false
  - Loop through each token (from 0 to tokco - 1)
    - If prog[i] is less than 0
      - Set x to true
      - Break the loop
  - Return x
```

Function win():

```
  - Initialize a boolean variable x to true
  - Loop through each token (from 0 to tokco - 1)
    - If prog[i] is not equal to 58
      - Set x to false
      - Break the loop
  - Return x
```

1. Initialize Static Members of the player Class:
   - Set player::pcount to 0
   - Set player::winp to 1

2. Initialize Thread Identifiers:
   - Declare an array tid of type pthread_t with 4 elements

3. Initialize Player Pointer:
   - Declare a pointer plid of type player and set it to 0 (nullptr)

4. Initialize Turns Pointer:
   - Declare a pointer turns of type int and set it to 0 (nullptr)

5. Initialize Semaphores:

```
  - Declare a semaphore di for controlling dice access
  - Declare a semaphore bo for controlling board access

Function Display Board(){

  1. Wait for a short period
     - Call plswait(0.5)

  2. Clear the screen
     - Call cls()

  3. Print the board layout
     - Print the top border of the board
     - Print each row of the board:
         - For each row, print the appropriate segments of the track and ltrack arrays with
colored backgrounds

     - The detailed print statements are:
        - Print the first row with parts of the track and ltrack arrays
        - Continue printing rows, adjusting for colors and positions
        - Include track and ltrack values with color codes

  4. Loop through each player and display their tokens
     - For i from 0 to plays-1:
        - Initialize offset coordinates for token display: offx = 5, offy = 14
        - Adjust offset coordinates based on player index:
           - If i > 1, set offy = 5
           - If (i + 1) & 2 is true, set offx = 25

        - Move the cursor to the offset position
           - Call printf with appropriate escape sequences to position the cursor at (offy, offx)

        - Loop through each token for the player and display it if not on the board:
           - For j from 0 to gotein-1:
              - If plid[i].prog[j] < 0, print the token:
                 - Print plid[i].oout with the appropriate color code for the player
                 - Use printf to add escape sequences for colors

        - Display the player's progress and positions:
           - Set offy to 5 + i
           - Call printf with escape sequences to position the cursor
           - Print the player's progress and positions:
              - For j from 0 to gotein-1:
                 - If plid exists, print plid[i].prog[j] and plid[i].pos[j]

  5. Move the cursor to a fixed position
     - Call printf with escape sequence to move cursor to (18, 1)
```

}

Function Player Turn:

```
1. Initialize player ID
   - id = (unsigned long)I

2. Wait for the dice semaphore
   - sem_wait(&di)

3. Initialize dice roll array
   - m[0], m[1], m[2] = 0, 0, 0

4. Roll the dice
   - Print "Player {id} has dice"
   - dice[0] = dicet[rand() % 8]
   - Print "Player {id} roll 1: {dice[0]}"
   - If dice[0] == 6:
      - dice[1] = dicet[rand() % 7]
      - Print "Player {id} roll 2: {dice[1]}"
      - If dice[1] == 6:
         - dice[2] = dicet[rand() % 6]
         - Print "Player {id} roll 3: {dice[2]}"
         - If dice[2] == 6:
            - Print "TOO MANY SIXES\nPASSING TURN FROM PLAYER {id}"
            - Reset dice: dice[0], dice[1], dice[2] = 0, 0, 0
            - Post the dice semaphore: sem_post(&di)
            - Exit the thread with return value id + 10: pthread_exit((void*)(id + 10))

5. Check if any tokens are open and the player didn't roll a 6
   - If not plid[id].anyopen() and dice[0] != 6:
      - Post the dice semaphore: sem_post(&di)
      - Exit the thread with return value id + 20: pthread_exit((void*)(id + 20))

6. Copy dice rolls to moves array
   - For i from 0 to 2:
      - m[i] = dice[i]
      - dice[i] = 0

7. Post the dice semaphore
   - sem_post(&di)

8. Wait for the board semaphore
   - sem_wait(&bo)
   - Print "Player {id} has board"

9. Initialize token to move
```

```
    - tok = -1

  10. Check if any tokens are open
    - If not plid[id].anyopen():
      - Print "None open for player {id}"

  11. Move tokens if possible
    - For i from 0 to 2:
      - If m[i] is non-zero and plid[id].canmove(m[i]) > -1:
      - tok = plid[id].canmove(m[i])
      - plid[id].mov(tok, m[i])

  12. Post the board semaphore
    - sem_post(&bo)

  13. Exit the thread with return value id
    - pthread_exit((void*)id)
```

Main Function()
{

```
  1. Initialize random seed
    - Call `srand(time(0))`

  2. Reset the game board and player tracks
    - For `i` from 0 to 51:
      - Call 'tracker set(i)`
    - For `i` from 0 to 3:
      - For `j` from 0 to 4:
        - Call `ltrackreset(i, j)`

  3. Clear the screen
    - Call `cls()`

  4. Set the number of players and tokens (hardcoded in this example)
    - Set `plays = 4`
    - Set `gotein = 2`

  5. Initialise semaphores
    - Call `sem_init(&di, 0, 0)`
    - Call `sem_init(&bo, 0, 0)`

  6. Display the initial board
    - Call `displayBoard()`

  7. Initialize player objects
    - Allocate memory for `plid` array of `player` objects: `plid = new player[plays]`
    - Set global pointer `p` to point to `plid`: `p = plid`
```

```
8. Initialize the round counter
   - Set `j = 0`

9. Post initial semaphore values to start the game
   - Call `sem_post(&di)`
   - Call `sem_post(&bo)`

10. Game loop for a fixed number of rounds (10 in this example)
   - While `j < 10`:
      - Print the start of a new round: `cout << "Round " << j << "\tTURN SEQUENCE:"`

      - Initialize turn sequence
         - Allocate memory for `turns` array: `turns = new int[plays]`
         - For `i` from 0 to `plays - 1`:
            - Set `turns[i] = i`

      - Shuffle the turn sequence
         - Call `random_shuffle(turns, turns + plays)`

      - Print the shuffled turn sequence
         - For `i` from 0 to `plays - 1`:
            - Print `turns[i]`

      - Create threads for each player's turn
         - For `i` from 0 to `plays - 1`:
            - Call `pthread_create(&tid[i], 0, playerTurn, (void*)turns[i])`

      - Wait for all threads to complete
         - For `i` from 0 to `plays - 1`:
            - Declare `void* tret`
            - Call `pthread_join(tid[i], &tret)`
            - Print thread result: `printf("\nmain:%i", (int*)tret)`

      - Display the updated board after all threads have completed
         - Call `displayBoard()`

      - Delete the turn sequence array
         - Call `delete[] turns`

      - Increment the round counter: `j++`

11. Clean up and deallocate resources
   - Delete the player objects array: `delete[] plid`
}
```

Ludo Code in C++:

```
#include <iostream>
#include <cstdlib>
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <ctime>
#include <algorithm>
#include <semaphore.h>
#include <string>

using namespace std;

void displayBoard(), *p = 0, stopcheck();
const int dicet[8] = {1, 6, 2, 3, 4, 5, 6, 6};
int dice[3], plays, gotein;
const  string  RESET_CLR  =  "\e[0m",  RED_CLR  =  "\e[31m",  GRN_CLR  =
"\e[32m", YELO_CLR = "\e[33m", BLU_CLR = "\e[34m", RED_BG = "\e[0;41m",
GRN_BG = "\e[0;42m", YELO_BG = "\e[0;43m", BLU_BG = "\e[0;44m", flash =
"\e[107m \e[0m";
string track[52], ltrack[4][5];
void  trackreset(int  i  =  0)  //  track[i][3]=background  by  Tens  digit;
track[i][5]= Ones value
{
     track[i] = "\e[40m0\e[0m", track[i][5] += i % 10, track[i][3] += (i
/ 10);
}
void  ltrackreset(int  i  =  0,  int  j  =  0)  //  track[i][j]  i  player,  jth
position on out track
{
    ltrack[i][j] = "\e[41m \e[0m", ltrack[i][j][3] += i;
}
inline void cls() { printf("\e[H\e[J"); }
inline void plswait(long double k)
{
    struct timespec remaining, request = {static_cast<long>(k),
                                          static_cast<long>(long((k -
(long)k) * 1000000000) % 1000000000)};
    nanosleep(&request, &remaining);
}

class player
{
```

```
public:
    bool beat = 0, active = 1;
    string oout, nam = "A";
    int id = 0, tokco = 4, offset = 52, *prog, *pos;
    player() : id(pcount++), tokco(gotein), oout("\e[0;1;91m#\e[0m")
    {
          prog = new int[tokco], pos = new int[tokco], oout[9] += id,
oout[7] += id;
        for (int i = 0; i < tokco; prog[i] = -1, pos[i++] = -1)
            ;
             offset -= (id + 1) * 13; // printf("Enter player %i
name:",id);cin>>nam;
        // cout<<oout<<endl;
    }
    int canmove(int n)
     { // return which token can move n steps; return token+=4 if an
opponent token is being wrecked
        int x = -1;
        if (!anyopen() and n != 6)
            return x; // none open
        if (anyclose() and n == 6)
            return aclosed();
        for (int i = 0; x < (4 * (rand() & 1)) && i < tokco; i++)
        {
            // open
            if (beat)
            {
                 if ((prog[i] + n) == 8 || (prog[i] == 51 && n == 1) ||
(prog[i] + n) == 13 || (prog[i] + n) == 21 || (prog[i] + n) == 26 ||
(prog[i] + n) == 34 || (prog[i] + n) == 39 || (prog[i] + n) == 47)
                {
                    x = i;
                    continue;
                } // on a stop
                for (int j = 0; j < tokco; j++)
                {
                    if (prog[i] != 51)
                    {
                        if (i != j && (prog[i] + n != prog[j]))
                        { // no overlap with friend tokens
                            x = i;
                            if (prog[i] + n < 51)
                            {
                                if (track[(pos[i] + n) % 52][2] == '4')
                                    ; // empty track
                                else
                                    x += 4; // opp on track (since it's
neither empty nor friend)
                            }
                            else
                                    ; // it's getting into the leaving
track;
                        }
                    }
                    else
                    {
                        if (i != j && ((prog[i] + n) % 52) != prog[j])
                        { // no overlap with friend tokens
                            x = i;
                            if (track[(pos[i] + n) % 52][2] == '4')
                                ; // empty track
                            else
                                x += 4; // opp on track
                        }
                    }
                }
            }
            else // no beat yet
            {
                 if ((prog[i] + n) == 8 || (prog[i] == 51 && n == 1) ||
(prog[i] + n) == 13 || (prog[i] + n) == 21 || (prog[i] + n) == 26 ||
(prog[i] + n) == 34 || (prog[i] + n) == 39 || (prog[i] + n) == 47)
                {
                    x = i;
                    continue;
                } // on a home color stop
                for (int j = 0; j < tokco; j++)
                {
                    if (i != j && ((prog[i] + n) % 52) != prog[j])
                    { // no overlap with friend tokens
                        x = i;
                        if (track[(pos[i] + n) % 52][2] == '4')
                            ; // empty track
                        else
                            x += 4; // opp on track
                    }
                }
            }
        }
    }
    int aclosed()
    {
        int x = -1;
        for (int i = 0; i < tokco; i++)
            if (prog[i] < 0)
                x = i;
        return x;
    }
    void bury(int p)
    {
        for (int i = 0; i < tokco; i++)
            if (pos[i] == p)
            {
                prog[i] = -1;
                track[pos[i]] = flash;
                displayBoard();
                trackreset(pos[i]);
                displayBoard();
                pos[i] = -1;
            }
    }
    void mov(int i, int n)
    {
        if (i < 4)
        {
            if (prog[i] < 0 && n == 6)
            {
                printf("\e[10%imToken opened\t\e[0m", id);
                prog[i] = 0, pos[i] = offset,
                track[pos[i]] = oout;
                return;
            }
            else if (beat)
            {
                if (prog[i] < 51)
                {
                    trackreset(pos[i]);
                    if ((prog[i] + n) > 50)
                    {
                        prog[i] += n + 1;
                        pos[i] = prog[i] % 52;
                        if (prog[i] == 57)
                            return;
                        ltrack[id][pos[i]] = oout;
                    }
                    else
                              prog[i] += n, pos[i] += n, pos[i] %= 52,
track[pos[i]] = oout;
                }
                else if (prog[i] == 51)
                {
                    trackreset(pos[i]);
                           prog[i] = n - 1, pos[i] = offset + n - 1,
track[pos[i]] = oout;
                }
                else
                {
                    ltrackreset(pos[i]), prog[i] += n, pos[i] += n;
                    if (prog[i] == 57)
                        return;
                    ltrack[id][pos[i]] = oout;
                }
            }
            else
            {
                trackreset(pos[i]);
                 prog[i] += n, prog[i] %= 52, pos[i] += n, pos[i] %= 52,
track[pos[i]] = oout;
            }
        }
        else
        {
            trackreset(pos[i]);
            int o = track[(pos[i] + n) % 52][7] - '1';
            ((player *)p)[o].bury((pos[i] + n) % 52);
            prog[i] += n, pos[i] += n, pos[i] %= 52, beat = 1;
            return;
        }
    }
    bool anyopen()
    {
        bool x = 0;
        for (int i = 0; i < tokco && !x; i++)
            x = (prog[i] >= 0 && prog[i] < 58);
        return x;
    }
    bool anyclose()
    {
        bool x = 0;
        for (int i = 0; i < tokco && !x; x = (prog[i++] < 0))
            ;
        return x;
    }
    bool win()
    {
        bool x = 1;
        for (int i = 0; i < tokco && x; x = (prog[i++] == 58))
            ;
        return x;
    }
    static int pcount, winp;
};
int player::pcount = 0;
int player::winp = 1;
pthread_t tid[4];
player *plid = 0;
int *turns = 0;
sem_t di, bo; // dice sem, board sem
void displayBoard()
{
    stopcheck();
    plswait(0.5);
    cls();
    cout << " _______________________________\n"
          << "|" << BLU_BG << "             \e[0m|" << track[10] << "|"
<<  track[11]  <<  "|"  <<  track[12]  <<  "|"  <<  YELO_BG  <<  "
\e[0m|\n"
            << "|" << BLU_BG << "               \e[0m|" << track[9] <<
YELO_BG << "|" << ltrack[2][0] << "|" << track[13] << "|" << YELO_BG <<
"            \e[0m|\n"
            << "|" << BLU_BG << "               \e[0m|" << track[8] <<
YELO_BG  <<  "|"  <<  ltrack[2][1]  <<  "|\e[0m"  <<  track[14]  <<  "|"  <<
YELO_BG << "             \e[0m|\n"
```

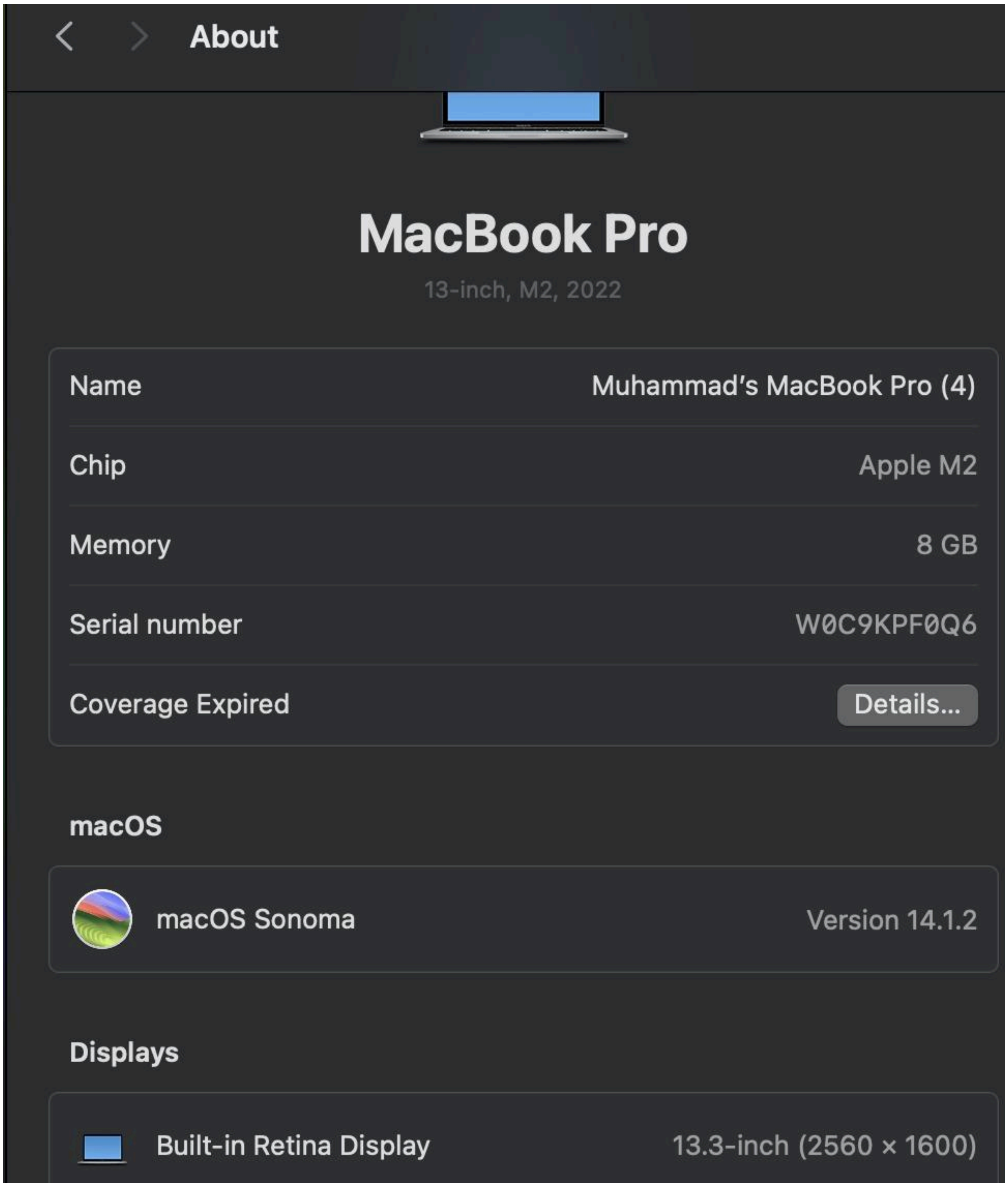
```
            << "|" << BLU_BG << "                \e[0m|" << track[7] <<
YELO_BG << "|" << ltrack[2][2] << "|\e[0m" << track[15] << "|" <<
YELO_BG << "             \e[0m|\n"
            << "|" << BLU_BG << "                \e[0m|" << track[6] <<
YELO_BG << "|" << ltrack[2][3] << "|\e[0m" << track[16] << "|" <<
YELO_BG << "             \e[0m|\n"
            << "|" << BLU_BG << "                \e[0m|" << track[5] <<
YELO_BG << "|" << ltrack[2][4] << "|\e[0m" << track[17] << "|" <<
YELO_BG << "             \e[0m|\n"
          << "|" << track[51] << BLU_BG << "|" << track[0] << "|\e[0m"
<< track[1] << "|" << track[2] << "|" << track[3] << "|" << track[4] <<
"|\e[100m          \e[0m|" << track[18] << "|" << track[19] << "|" <<
track[20] << "|" << track[21] << "|" << track[22] << "|" << track[23]
<< "|\n"
          << "|" << track[50] << BLU_BG << "|" << ltrack[3][0] << "|" <<
ltrack[3][1] << "|" << ltrack[3][2] << "|" << ltrack[3][3] << "|" <<
ltrack[3][4] << "|\e[0;100m              \e[0m" << GRN_BG << "|" <<
ltrack[1][4] << "|" << ltrack[1][3] << "|" << ltrack[1][2] << "|" <<
ltrack[1][1] << "|" << ltrack[1][0] << "|\e[0m" << track[24] << "|\n"
          << "|" << track[49] << "|" << track[48] << "|" << track[47] <<
"|" << track[46] << "|" << track[45] << "|" << track[44] << "|\e[0;100m
\e[0m|" << track[30] << "|" << track[29] << "|" << track[28] << "|" <<
track[27] << GRN_BG << "|" << track[26] << "|\e[0m" << track[25] <<
"|\n"
            << "|" << RED_BG << "                \e[0m|" << track[43] <<
RED_BG << "|" << ltrack[0][4] << "|\e[0m" << track[31] << "|" << GRN_BG
<< "             \e[0m|\n"
            << "|" << RED_BG << "                \e[0m|" << track[42] <<
RED_BG << "|" << ltrack[0][3] << "|\e[0m" << track[32] << "|" << GRN_BG
<< "             \e[0m|\n"
            << "|" << RED_BG << "                \e[0m|" << track[41] <<
RED_BG << "|" << ltrack[0][2] << "|\e[0m" << track[33] << "|" << GRN_BG
<< "             \e[0m|\n"
            << "|" << RED_BG << "                \e[0m|" << track[40] <<
RED_BG << "|" << ltrack[0][1] << "|\e[0m" << track[34] << "|" << GRN_BG
<< "             \e[0m|\n"
          << "|" << RED_BG << "              \e[0m|" << track[39] << "|"
<< ltrack[0][0] << "|\e[0m" << track[35] << "|" << GRN_BG << "
\e[0m|\n"
          << "|" << RED_BG << "              \e[0m|" << track[38] << "|"
<< track[37] << "|" << track[36] << "|" << GRN_BG << "
\e[0m|\n"
         << " --------------------------------\n";
```

```
    // for(int i=1;i<45;printf("%i",i++%10));

    for (int i = 0; i < plays; i++)
    {
        int offx = 5, offy = 14;
        if (i > 1)
            offy = 5;
        if ((i + 1) & 2)
            offx = 25;
        printf("\e[%i;%iH", offy, offx);
        for (int j = 0; j < gotein; j++)
            if (plid && (plid[i].prog[j] < 0))
                cout << plid[i].oout << "\e[4" << char('1' + i) << "m";
        offy = 5 + i;
        printf("\e[%i;46H\e[10%improg,pos %i:", offy, i + 1, i);
        for (int j = 0; j < gotein; j++)
            if (plid)
                printf("(%i,%i) ", plid[i].prog[j], plid[i].pos[j]);
    }
    printf("\e[18;1H");
}
void *playerTurn(void *I)
{
    unsigned long id = (unsigned long)I;
    sem_wait(&di);
    int m[3] = {0};
    printf("\e[0;3%imPlayer %i has dice\t", id + 1, id);
    plswait(1);
    dice[0] = dicet[rand() % 8];
    printf("\e[0;3%imPlayer %i roll 1:%i||", id + 1, id, dice[0]);
    if (dice[0] == 6)
    {
        dice[1] = dicet[rand() % 7];
        plswait(1);
        printf("\e[0;3%imPlayer %i roll 2:%i||", id + 1, id, dice[1]);
        if (dice[1] == 6)
        {
            dice[2] = dicet[rand() % 6];
            plswait(1);
             printf("\e[0;3%imPlayer %i roll 3:%i\t\e[0m\n", id + 1, id,
dice[2]);
            if (dice[2] == 6)
            {
                    printf("\e[0;1;91mTOO MANY SIXES\nPASSING TURN FROM
PLAYER %i\n\e[0m", id);
                dice[0] = 0, dice[1] = 0, dice[2] = 0;
                sem_post(&di); // printf("\nKKKKK%ul",id);
                pthread_exit((void *)(id + 10));
            }
        }
    }
    if (!plid[id].anyopen() && dice[0] != 6)
    {
        sem_post(&di);
        pthread_exit((void *)(id + 20));
    }
    for (int i = 0; i < 3; m[i] = dice[i], dice[i++] = 0)
        ; // copying moves of dice to this player's turn's moves array
    sem_post(&di);
    sem_wait(&bo);
    printf("\e[0;3%imPlayer %i has board\t\e[0m", id + 1, id);
    ////////////////////////movement management
    ///////////////////////////////////////////////
    int tok = -1;
    if (!plid[id].anyopen())
        printf("\e[10;46HNone open of %i\e[19;1H", id);
    // some are open
    for (int i = 0; m[i] && i < 3; i++)
        if (plid[id].canmove(m[i]) > -1)
        {
            tok = plid[id].canmove(m[i]);
            plid[id].mov(tok, m[i]);
        }

    sem_post(&bo);
    pthread_exit((void *)id);
}

int main()
{
    srand(time(0));
    char y = 'y';
    for (int i = 0; i < 52; trackreset(i++))
         ; // track[i][3]=background by Tens digit; track[i][5]= Ones
value
    for (int i = 0; i < 4; i++)
```

```
        for (int j = 0; j < 5; ltrackreset(i, j++))
            ;
    cls();
        printf("\e[1;36;104mStarting  Ludo!!!!\n\n\e[0mEnter  number  of
players:\t"); /*cin>>plays; plays%=5;
     printf("Enter number of tokens:\t");cin>>gotein;
        if(plays<2  ||  gotein<1)  {printf("\e[1;101mWāh,  kyā  kamāl  kām
kīyā.. jītē reh...\n\e[0m"); return 100;}
     */
    plays = 4, gotein = 2;
    sem_init(&di, 0, 0);
    sem_init(&bo, 0, 0);
    displayBoard();
    plid = new player[plays];
    p = plid;
    int j = 0;
    sem_post(&di);
    sem_post(&bo);
    while (y == 'y')
    {
        // all players wait
        turns = new int[plays];
        for (int i = 0; i < plays; turns[i++] = i)
            ;
        random_shuffle(turns, turns + plays);
        void *tret;
        cout << "\e[0mRound " << j++ << "\tTURN SEQUENCE:";
        for (int i = 0; i < plays; cout << turns[i++] << ' ')
            ;
        cout << endl;
        for (int i = 0; i < plays; i++)
            pthread_create(&tid[i], 0, playerTurn, (void *)turns[i]);
                        for    (int    i    =    0;    i    <    plays;
/*printf("\e[0m\nmain:%i",(int*)tret)*/)
            pthread_join(tid[i++], &tret);

        displayBoard();
        delete[] turns;
        printf("Round %i, Enter y to continue to next round ", j);
        cin >> y;
    }
    delete[] plid;
}
```

```
void stopcheck()
{
    int stops[8] = {8, 0, 13, 21, 26, 34, 39, 47};
    for (int s = 0; s < 8; s++)
    {
        bool x = 0;
        if (plid)
            for (int i = 0; !x && i < plays; i++)
                for (int j = 0; !x && j < gotein; j++)
                    if (plid[i].pos[j] == stops[s])
                        track[plid[i].pos[j]] = plid[i].oout, x = 1;
    }
}
```

System Specifications:



About

MacBook Pro

13-inch, M2, 2022

Name	Muhammad's MacBook Pro (4)
Chip	Apple M2
Memory	8 GB
Serial number	W0C9KPF0Q6
Coverage Expired	Details...

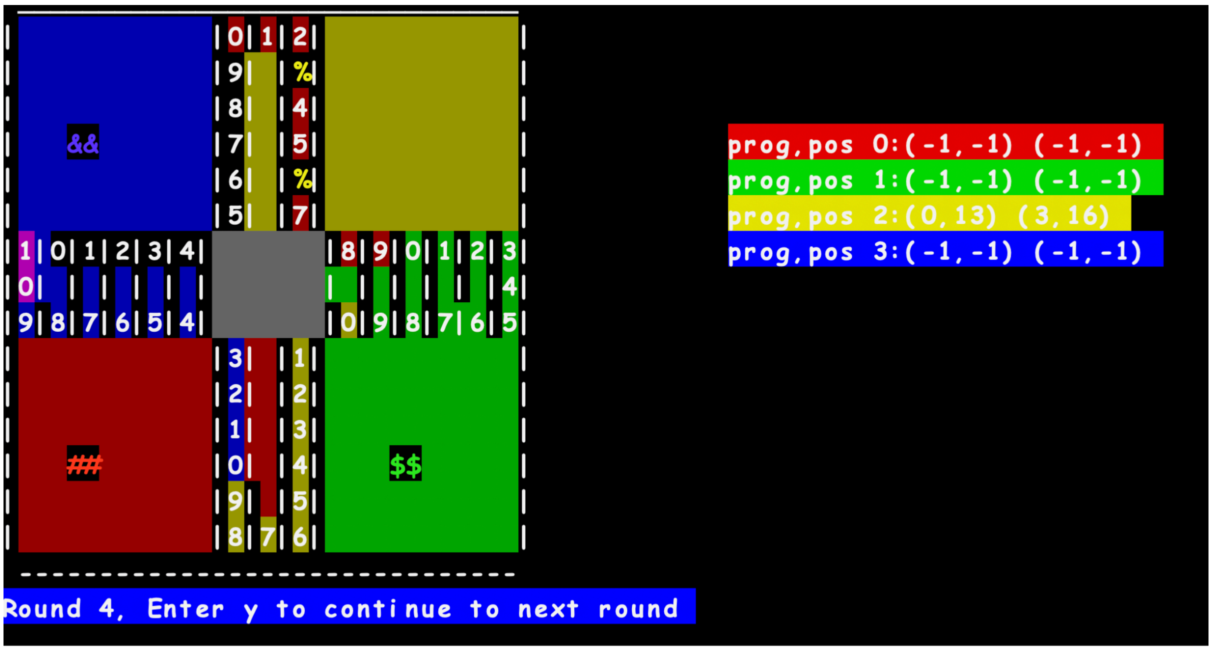
macOS

macOS Sonoma Version 14.1.2

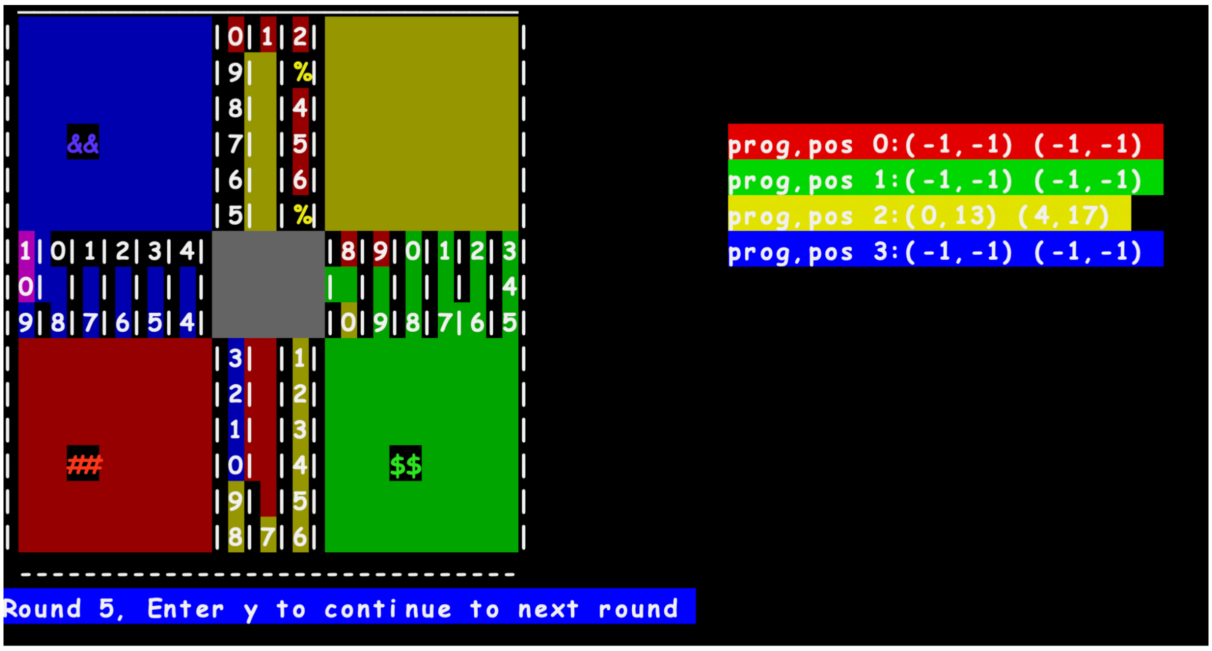
Displays

Built-in Retina Display 13.3-inch (2560 × 1600)

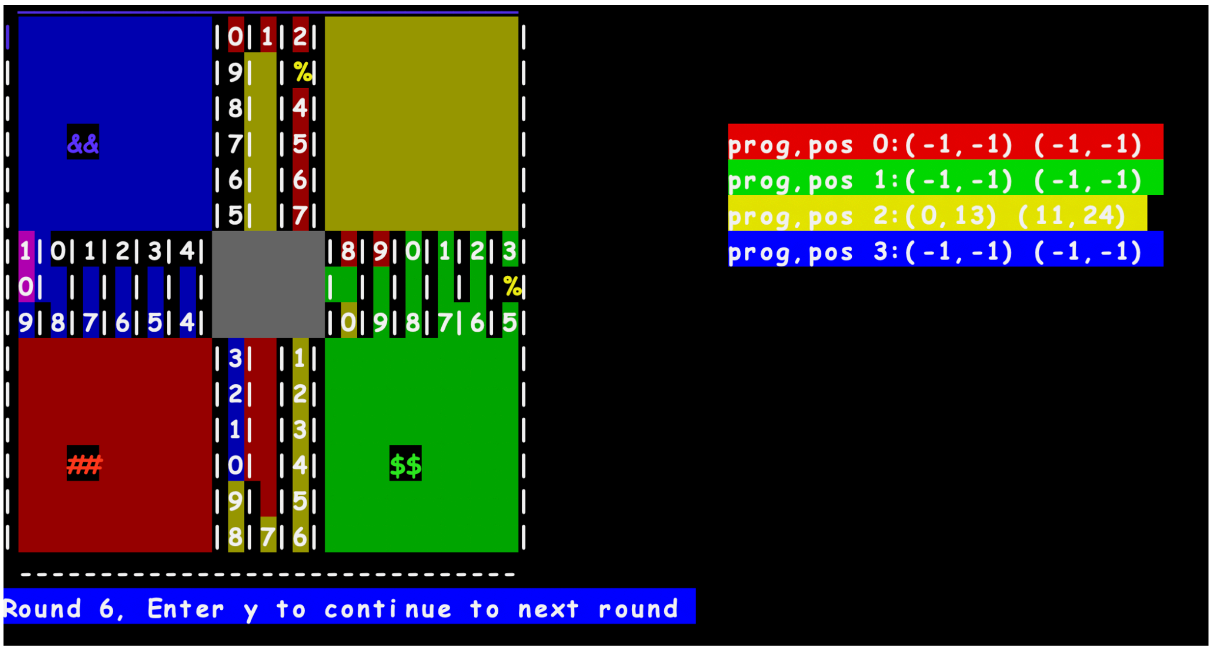
Illustrations of our Code:



```
 _____________________________
|           |0|1|2|           |
|           |9| |%|           |
|           |8| |4|           |
|    &&     |7| |5|           |
|           |6| |%|           |
|           |5| |7|           |
|1|0|1|2|3|4|     |8|9|0|1|2|3|
|0| | | | | |     | | | | | |4|
|9|8|7|6|5|4|     |0|9|8|7|6|5|
|           |3| |1|           |
|           |2| |2|           |
|           |1| |3|           |
|    ##     |0| |4|    $$     |
|           |9| |5|           |
|           |8|7|6|           |
 -----------------------------
Round 4, Enter y to continue to next round

prog,pos 0:(-1,-1) (-1,-1)
prog,pos 1:(-1,-1) (-1,-1)
prog,pos 2:(0,13) (3,16)
prog,pos 3:(-1,-1) (-1,-1)
```



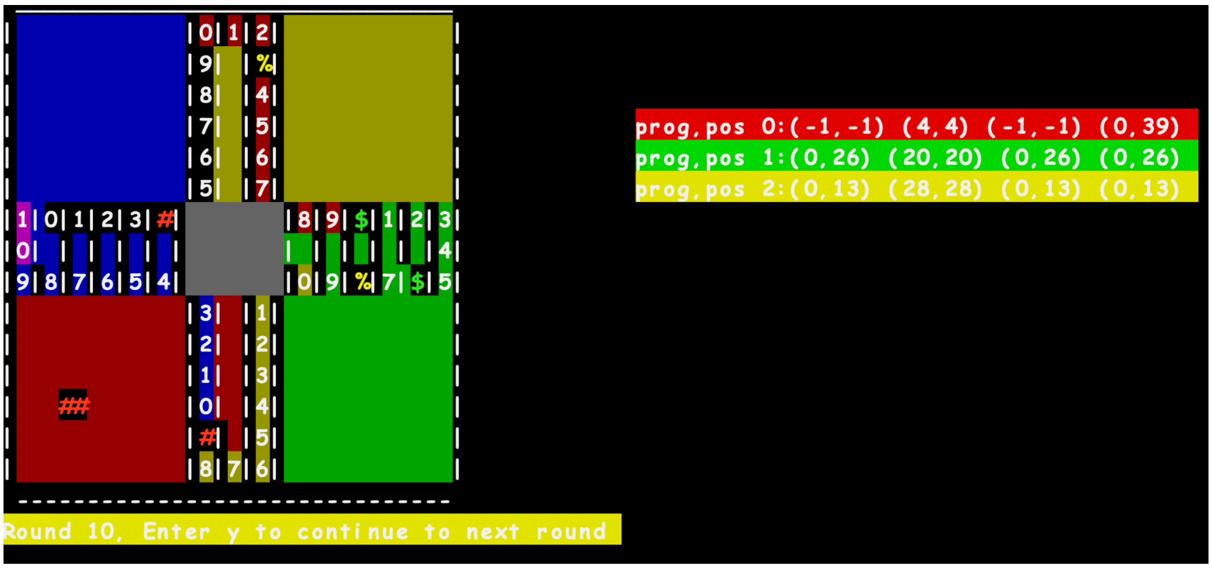
```
 _____________________________
|           |0|1|2|           |
|           |9| |%|           |
|           |8| |4|           |
|    &&     |7| |5|           |
|           |6| |6|           |
|           |5| |%|           |
|1|0|1|2|3|4|     |8|9|0|1|2|3|
|0| | | | | |     | | | | | |4|
|9|8|7|6|5|4|     |0|9|8|7|6|5|
|           |3| |1|           |
|           |2| |2|           |
|           |1| |3|           |
|    ##     |0| |4|    $$     |
|           |9| |5|           |
|           |8|7|6|           |
 -----------------------------
Round 5, Enter y to continue to next round

prog,pos 0:(-1,-1) (-1,-1)
prog,pos 1:(-1,-1) (-1,-1)
prog,pos 2:(0,13) (4,17)
prog,pos 3:(-1,-1) (-1,-1)
```



```
prog,pos 0:(-1,-1) (-1,-1)
prog,pos 1:(-1,-1) (-1,-1)
prog,pos 2:(0,13) (11,24)
prog,pos 3:(-1,-1) (-1,-1)
```

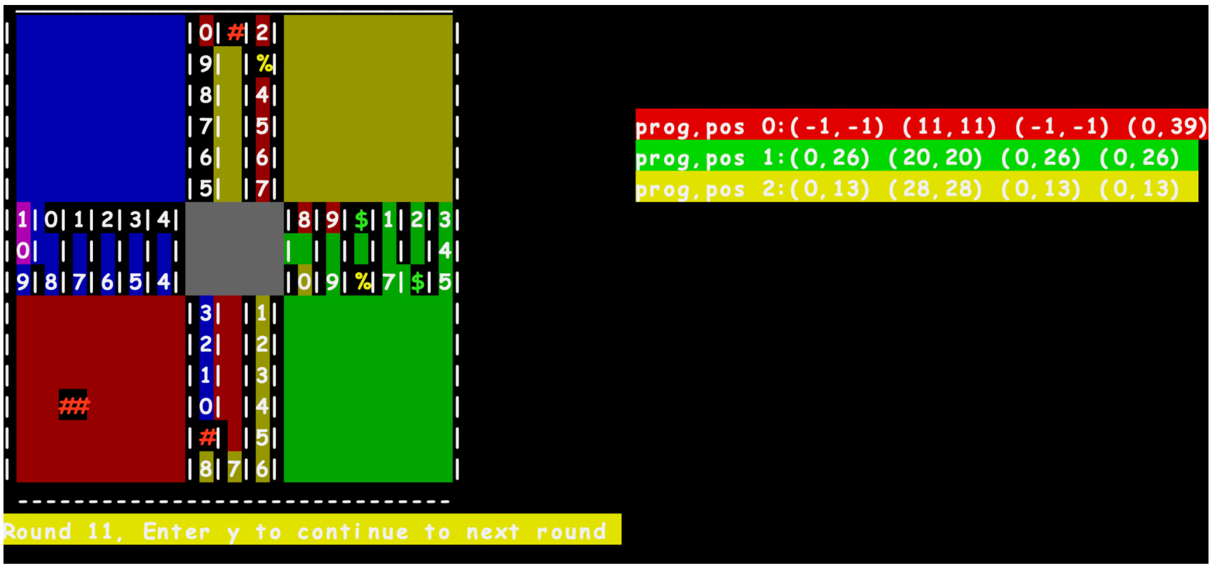
```
--------------------------------
Round 6, Enter y to continue to next round
```

3 players and 4 tokens



```
prog,pos 0:(-1,-1) (4,4) (-1,-1) (0,39)
prog,pos 1:(0,26) (20,20) (0,26) (0,26)
prog,pos 2:(0,13) (28,28) (0,13) (0,13)
```

```
--------------------------------
Round 10, Enter y to continue to next round
```



```
prog,pos 0:(-1,-1) (11,11) (-1,-1) (0,39)
prog,pos 1:(0,26) (20,20) (0,26) (0,26)
prog,pos 2:(0,13) (28,28) (0,13) (0,13)

Round 11, Enter y to continue to next round
```

Paragraph to implement concepts in other scenario:

We can implement these concepts for **A Multi-Threaded Restaurant Simulation**

Context:
We can simulate a restaurant where multiple waiters (threads) serve customers (tasks). The restaurant has limited resources like tables and kitchen capacity, requiring synchronised access.

Implementation Details:

Synchronisation with Semaphores:

We can use semaphores to manage access to shared resources like tables and kitchen slots.
For example, sem_t tables to track available tables and sem_t kitchen for kitchen capacity.

Multi-Threading with pthreads:

Each waiter can operate as a separate thread (pthread_t waiter_threads[NUM_WAITERS]).
Threads execute tasks concurrently, representing waiters serving customers simultaneously.

Task Allocation and Management:

Customers can arrive randomly, and each waiter thread picks up customer orders.
Similar to the playerTurn function, a waiterTask function handles the sequence of actions (taking orders, serving food, processing payments).

Game Logic:

The game's dice roll logic can be compared to random customer arrivals and order assignments.

Token movement logic corresponds to waiters moving between tables and the kitchen, with conditions to check if actions can be performed (e.g., if a table is available).

Resource Management:

Just as the game checks if a token can move, the simulation checks if resources (tables, kitchen slots) are available before proceeding.
Use similar logic to handle resource state changes and updates to shared data structures.

This approach demonstrates how concepts from the Ludo game implementation can be adapted to model a multi-threaded restaurant simulation effectively.